User Guide for Stochastic Radial Basis Function Algorithm
Python Version

Juliane Müller
Haoyu Jia
email: juliane.mueller2901@gmail.com
hj342@cornell.edu

January 28, 2014

1 Introduction

This user guide accompanies the stochastic radial basis function (RBF) algorithm for global optimization problems. The algorithm attempts to find accurate solutions for minimization problems of the following form:

$$\min f(\mathbf{x}), \qquad \text{subject to } x_i^l \leq x_i \leq x_i^u,\ i = 1, \ldots, d, \tag{1}$$

where $f(\mathbf{x})$ is a computationally expensive objective function (often a time consuming simulation model) whose analytical description is not available (black box). Considered are box-constrained optimization problems, i.e. only lower (x_i^l) and upper (x_i^u) variable bounds exist for $x_i \in \mathbb{R},\ i = 1, \ldots, d$, where d is the problem dimension. There are no other constraints.

Note that for problems with computationally cheap function evaluations the algorithm may not be very efficient. Surrogate models are intended to be used when a single function evaluation takes from several minutes to several hours or more. When reading this manual it is recommended to simultaneously take a look at the code and to try out the examples. It is assumed that the user is familiar with the papers on whose content this implementation is based:

- A Stochastic Radial Basis Function Method for the Global Optimization of Expensive Functions by R.G. Regis and C.A. Shoemaker, 2007, *INFORMS Journal on Computing*, vol. 19, pp. 497-509
- Parallel Stochastic Global Optimization Using Radial Basis Functions by R.G. Regis and C.A. Shoemaker, 2009, *INFORMS Journal on Computing*, vol. 21, pp. 411-426

These papers should be cited and this Python implementation should be referenced whenever the codes are used to generate results for the user's own research. The user is urged to read these papers before continuing with the manual since it helps understanding the following descriptions.

There is a Matlab implementation of this Python code available on Matlab File Exchange that is based on the implementation by R.G. Regis and Y. Wang and has been edited by J. Müller. See also R.G. Regis' webpage `http://people.sju.edu/~rregis/pages/software.html` for the original code. The Python implementation you just downloaded contains the option for doing several function evaluations in parallel (in addition to the option of doing one evaluation at a time).

The code is set up such that the user only has to define his/her optimization problem in a Python file (see Section 6.1). Additional input such as the maximum number of allowed function evaluations, the number of trials, an indication if the results should be plotted, and the number of function evaluations to be done in every iteration are optional, and if not given by the user, default values are assigned (see Section 6).

This document is structured as follows. In Section 2 the general surrogate model algorithm is described. The installation of the algorithm is described in Section 3. The dependencies of the single functions in the code are shown in Section 4. Section 5 briefly describes the main function StochasticRBF.py. Section 6 describes the options for the input arguments of the main function and contains an example. The elements of the saved results are described in Section 7.

Finally, if you have any questions and recommendations, or if you encounter any bugs, please feel free to contact me at the email address `juliane.mueller2901@gmail.com`.

2 Surrogate Model Algorithms

Surrogate models (also known as response surfaces or metamodels) are used in the optimization algorithms to approximate expensive simulation models [1]. During the optimization phase information from the surrogate model is used in order to guide the search for improved solutions. Using the surrogate model instead of the true simulation model reduces the computation time considerably. Most surrogate model algorithms consist of the same steps as shown in the algorithm below.

Algorithm ***General Surrogate model Algorithm***

1. Generate an initial experimental design.
2. Do the costly function evaluations at the points generated in Step 1.
3. Fit a response surface to the data generated in Steps 1 and 2.
4. Use the response surface to predict the objective function values at unsampled points in the variable domain to decide where to do the next expensive function evaluation.
5. Do the expensive function evaluation at the point(s) selected in Step 4.
6. Use the new data point(s) to update the surrogate model.
7. Iterate through Steps 4 to 6 until the stopping criterion has been met.

Typically used stopping criteria are a maximum number of allowed function evaluations (adopted in this implementation), a maximum allowed CPU time, or a maximum number of failed iterative improvement trials.

3 Installation

Required Enviroment:

- Python interpreter 2.7.3 (`http://www.python.org/download/releases/2.7.3/`)
- NumPy 1.8.0 (`http://www.scipy.org/Download`)
- matplotlab 1.2.0 (`http://matplotlib.org/`)

Note: matplotlab 1.2.0 may require NumPy version above 1.5.0

Download the file StochasticRBF_py.zip and unzip it. Open terminal and change directory to the StochasticRBF_py folder. You can try to run StochasticRBF as a demo script. In the command prompt, type

```
$: python StochasticRBF
```

To run the program as a function call, you can try this:

```
$: python
>>> from StochasticRBF import StochasticRBF
>>> StochasticRBF('datainput_hartman3',200,3,1,1)
```

You can also use it as a function in your own program by importing the *StochasticRBF* module and calling the *StochasticRBF* function in the same way as above.

4 Code Structure

The structure of the code is outlined here. The module at the highest level (StochasticRBF.py) is the function that has to be called by the user. The subtrees indicate dependencies between the subfunctions.

```
StochasticRBF.py
└─TestLocalStochRBFrestart.py
  └─LocalStochRBFrestart.py
    ├─SLHDstandard.py
    └─LocalStochRBFstop.py
      ├─InitialRBFMatrices.py
      │ └─phi.py
      ├─Minimize_Merit_Function.py
      │ └─ComputeRBF.py
      │   └─phi.py
      └─phi.py
```

One important module not included in the above tree is utility.py. The module utility.py is imported by every module above and depends only on NumPy. Three important data structures, myException, Data, Solution, are defined in utility.py. The module utility.py is highly recommended to look at for the user when defining an own optimization problem. Alternatively, look at the example datainput_hartman3.py for an example of how to define a problem.

5 The Main File **StochasticRBF.py**

The module from which to run the algorithm is StochasticRBF.py. The file expects several inputs (see Section 6) of which only the first one is mandatory. The algorithm starts by checking if the input is correct and assigns default values to variables that have not been set by the user. If any mandatory input data is missing or incorrect, the algorithm terminates with an error message indicating where the problem may be. Parameters, such as the type of the used RBF model, the corresponding polynomial tail, and the number of candidate points, are set. After the optimization finished, a plot of the results is generated (if so desired by the user). The algorithm saves the results in the file Results.data.

6 Input

The main file StochasticRBF.py requires several input arguments:

StochasticRBF(data_file, maxeval, Ntrials, PlotResult, NumberNewSamples)

See Table 1) for details. Only the first argument is mandatory to run the algorithm. If no input is given for the remaining arguments, default values are used.

Table 1: Input parameters

Input	Description
data_file	string with name of file containing optimization problem data (mandatory!)
maxeval	positive integer defining maximum number of allowed function evaluations (default $20 \cdot d$, d =dimension)
Ntrials	positive integer defining the number of times the algorithm is executed for the given problem (default 1)
PlotResult	0 = no plot; 1 = plot (default 1)
NumberNewSamples	positive integer defining the number of points selected in every iteration of the algorithm for doing expensive simulation (default 1)

6.1 Input data_file

The data file contains all the necessary problem information. See for example the file datainput_hartman3.py. The data file must define a function with the same name as the data file. This function has no input argument, and one output argument (the structure variable Data). An object of the Data structure must be defined and must contain the information shown on Table 2. You can also refer to utility.py to find the required fields in the Data object. The Data structure also provides a function validation to check whether the user-given dimension and the lower/upper bounds are valid.

Table 2: Contents of data_file

Variable	Description
Data.xlow	variable lower bounds, row vector with d (=dimension) entries
Data.xup	variable upper bounds, row vector with d (=dimension) entries
Data.dim	problem dimension, positive integer
Data.objfunction	handle to objective function/simulation model, must return a scalar value

6.2 Input Ntrials

The input Ntrials indicates how often StochasticRBF should be run for the same problem. The reason for running the algorithm more than once for the same problem is the random component when creating the initial experimental design and when generating candidate points. In order to average out the effect of these random components, several trials should be made. However, for computationally expensive problems this might not be possible due to the required computation time for doing the expensive function evaluations. Hence, for most application problems, Ntrials =1 is a reasonable choice.

6.3 Input PlotResult

If set to 1 (or any value different from 0), a plot of the best objective function value averaged over all trials after a given number of function evaluations is made. This allows the user to see the progress of the algorithm and assess convergence.

6.4 Input NumberNewSamples

The variable NumberNewSamples indicates how many points are to be selected in every iteration of the algorithm for doing expensive function evaluations. If NumberNewSamples is larger than one, then the function evaluations are done in parallel (as suggested in [3]). Note that the objective function values for the points in the initial experimental design are in this implementation computed iteratively.

6.5 Input Example

The following example executes the stochastic RBF algorithm for finding the minimum of the three-dimensional Hartmann function defined in the file datainput_hartman3.py. The maximum number of function evaluations is set to 200, Ntrials is set to 3 (the algorithm is started 3 times for the problem, and each trial has a different seed for the random number generator). PlotResult is set to 1 in order to illustrate the development of the objective function value vs. the number of function evaluations, and NumberNewSamples is set to 2, i.e. in every iteration two new points are selected and the objective function values of these two points are computed simultaneously. The user is encouraged to try out the example by typing into the python command prompt (make sure the current directory is in the folder):

```
>> StochasticRBF('datainput_hartman3',200,3,1,2)
```

Note that in the command window the iteration number and the number of function evaluations done so far is shown. The plot of the average objective function value vs. the number of function evaluations should look similar to the graph in Figure 1.

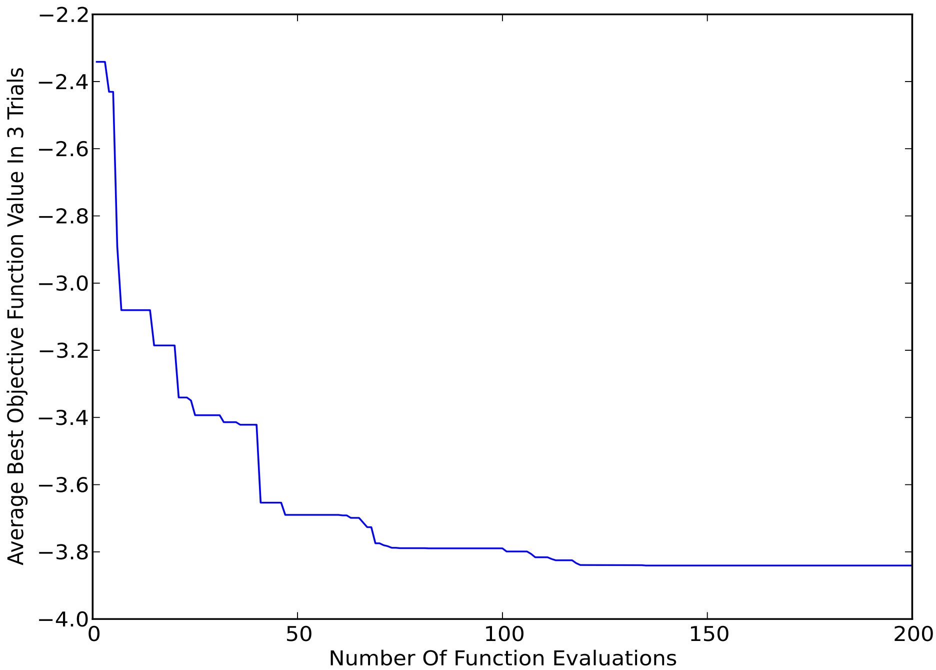

Figure 1: Average objective function value vs. number of function evaluations.

7 Results

The algorithm saves the results of the optimization to the file Result.data. The results are written to the file by cPickle in Python standard library, refer to `http://docs.python.org/2/library/pickle.html` if you are not familiar with python Pickle. cPickle uses exactly the same interface as Pickle, but is implemented in language C, which makes it faster than Pickle.

To load the results to your python program, you can also use cPickle. In python command prompt or your own program:

```
>> import cPickle as p
>> Solution = p.load(open('Result.data'))
```

If the algorithm has not been interrupted (e.g. by pressing CTRL+C), the following elements are contained in the saved Solution structure (see Table 3). You can also refer to utility.py to find the elements of the Solution structure.

Table 3: Saved Solution structure elements

Elements	Description
Solution.BestPoints	(Ntrials$\times d$) matrix with best point found in each trial of the algorithm
Solution.BestValues	(Ntrials$\times 1$) matrix with best objective function value found in each trial of the algorithm
Solution.NumFuncEval	(Ntrials$\times 1$) matrix with number of function evaluations in each trial
Solution.AvgFuncEvalTime	(Ntrials$\times 1$) matrix with average time needed for evaluating the objective function in each trial
Solution.FuncVal	(maxeval $\times$ Ntrials) matrix with objective function values in every trial (ith column corresponds to ith trial)
Solution.DMatrix	(maxeval $\times d \times$ Ntrials) matrix with points where objective function has been evaluated in each trial. Third dimension corresponds to trial number
Solution.NumberOfRestarts	(Ntrials $\times$ 1) matrix with number of optimization restarts in each trial. The optimization reboots whenever a local optimum has been encountered and if there is a budget of function evaluations left.

8 Exception Handling

All the errors in this program are handled by raising an object of the structure myException. You can refer to utility.py for the declaration and definition of the myException structure. The data structure contains one string member named msg as message. All the exceptions in this program are handled in StochasticRBF from StochasticRBF.py, and messages are printed to the command prompt. You can also use myException in your program by importing the utility module.

9 GNU Free Documentation License

References

[1] A.J. Booker, J.E. Dennis Jr, P.D. Frank, D.B. Serafini, V. Torczon, and M.W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Multidisciplinary Optimization*, 17:1–13, 1999.

[2] R.G. Regis and C.A. Shoemaker. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19:497–509, 2007.

[3] R.G. Regis and C.A. Shoemaker. Parallel stochastic global optimization using radial basis functions. *INFORMS Journal on Computing*, 21:411–426, 2009.